```
# If running in Colab, you can run these installation commands.
# Otherwise, you can skip if your environment is already set up.

# Mount Google Drive (if using Colab)
from google.colab import drive
drive.mount('/content/drive')

!pip install --upgrade pip
!pip install PyPDF2 pdfplumber pytesseract spacy openai pandas openpyxl
!pip install pydrive transformers

# If you specifically need openai==0.28, uncomment below:
!pip install openai==0.28

!apt-get install -y tesseract-ocr
!pip install pytesseract
!python -m spacy download en_core_web_sm
```

```
Setting up tesseract-ocr-eng (1:4.00~git30-7274cfa-1.1) ...
Setting up tesseract-ocr-osd (1:4.00~git30-7274cfa-1.1) ...
Setting up tesseract-ocr (4.1.1-2.1build1) ...
Processing triggers for man-db (2.10.2-1) ...
Requirement already satisfied: pytesseract in /usr/local/lib/python3.11/dist-packages (0.3.13)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from pytesseract) (24.2)
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from pytesseract) (11.1.0)
Collecting en-core-web-sm==3.7.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.7.1/en_core_web_sm-3.7.1-py3-none-any.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 91.2 MB/s eta 0:00:00
Requirement already satisfied: spacy<3.8.0,>=3.7.2 in /usr/local/lib/python3.11/dist-packages (from en-core-web-sm==3.7.1) (3.7.5)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-c
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-c
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-cor
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-we
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.11/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==3.
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy<3.8
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>
Requirement already satisfied: pydantic-core==2.27.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spa
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.8.0,>=3
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.8
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.8
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.11/dist-packages (from thinc<8.3.0,>=8.2.2->spacy<3.8.0,>
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.3.0,>=8.2.2->spacy<3
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.2
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spac
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spacy<
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->spacy<3.8.0,>=3.7.2->en-core-
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from language-data>=1.2->langcodes<4.0.
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>=
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0-
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->type
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

```
import os
import PyPDF2
import pdfplumber
```

```
from pytesseract import pytesseract
from PIL import Image
import spacy
import re
import openai
import pandas as pd
import json
import time
import hashlib
import queue
from collections import deque

# Load spaCy model
nlp = spacy.load('en_core_web_sm')

# Set your (new, secure!) OpenAI API Key
openai.api_key = "sk-proj-aRc9bay31-wbbfw9jyodxYTqXiQ2s6K6bDiO-2Hz02gtd3qYxjnCFo6lqpF-q37SxVSJGSweA8T3BlbkFJtTKoQGATUqSOr4a5dhxVCw14WZFfT1gc

print("Environment setup complete.")
```

```
Environment setup complete.
```

```
resume_folder = "/content/drive/MyDrive/Generative AI-Powered Resume Analyzer/Resume/"
resumes = [f for f in os.listdir(resume_folder) if f.endswith('.pdf')]
print(f"Found {len(resumes)} resumes in '{resume_folder}':")
print(resumes)
```

```
Found 13 resumes in '/content/drive/MyDrive/Generative AI-Powered Resume Analyzer/Resume/':
['Swapnil_22MCES16.pdf', 'Prathamesh Bute_Resume.pdf', 'Manan_Patel_Resume.pdf', '14049846.pdf', '15601399.pdf', '13264796.pdf', '151399
```

```
def extract_text_from_pdf(pdf_path):
    """Extract text from text-based PDF using PyPDF2."""
    try:
        with open(pdf_path, 'rb') as file:
            reader = PyPDF2.PdfReader(file)
            text = ""
            for page in reader.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text + "\n"
            return text
    except Exception as e:
        print(f"Error reading {pdf_path} with PyPDF2: {e}")
        return ""

def extract_text_with_ocr(pdf_path):
    """Extract text using pdfplumber; fallback to OCR if page has no text."""
    try:
        text = ""
        with pdfplumber.open(pdf_path) as pdf:
            for page in pdf.pages:
                page_text = page.extract_text()
                if page_text and page_text.strip():
                    text += page_text + "\n"
                else:
                    img = page.to_image().to_pil()
                    text += pytesseract.image_to_string(img) + "\n"
        return text
    except Exception as e:
        print(f"Error processing {pdf_path} with pdfplumber/pytesseract: {e}")
        return ""


resume_texts = {}
for resume_file in resumes:
    pdf_path = os.path.join(resume_folder, resume_file)
    # First try PyPDF2
    basic_text = extract_text_from_pdf(pdf_path)
    if basic_text.strip():
        resume_texts[resume_file] = basic_text
    else:
        # Fallback to OCR
        text_ocr = extract_text_with_ocr(pdf_path)
        resume_texts[resume_file] = text_ocr
```

```
print(f"Extracted text for {len(resume_texts)} resumes.")
```

```
Extracted text for 13 resumes.
```

```
# We'll store a dictionary of {hash_of_text: extracted_data}
# So we don't call GPT repeatedly on the same text.

cache_dict = {}

def get_text_hash(text):
    return hashlib.sha256(text.encode('utf-8')).hexdigest()


def format_phone_number(raw_phone):
    """
    Normalizes phone numbers according to these rules:
    1) Keep digits + leading plus sign if present.
    2) If it starts with +91 and has exactly 13 total characters (e.g. +911234567890),
       format as +91-XXXXXXXXXX (e.g. +91-1234567890).
    3) If it's exactly 10 digits (e.g. 7016151785), format as (701) 615-1785.
    4) Otherwise, if it starts with +91- already, keep it as is.
    5) Else, just return the raw phone with minimal cleanup.
    """

    # Extract only plus sign (if at start) and digits
    match = re.match(r"(\+?\d+)", raw_phone)
    if not match:
        return raw_phone  # If nothing valid, return as is

    phone = match.group(1)

    # Rule 2: +91 plus 10 digits => total length = 13
    # Example: +911234567890 => +91-1234567890
    if phone.startswith("+91") and len(phone) == 13:
        # Example: +91 1234567890
        prefix = phone[:3]  # +91
        rest   = phone[3:]  # 1234567890
        return f"{prefix}-{rest}"

    # Rule 3: If exactly 10 digits => (701) 615-1785 style
    if len(phone) == 10 and phone.isdigit():
        return f"({phone[0:3]}) {phone[3:6]}-{phone[6:]}"

    # Rule 4: If it already starts with +91- (like +91-7016151785), keep it
    if phone.startswith("+91-"):
        return phone

    # Rule 5: Fallback => Return raw phone with minimal dashes
    if len(phone) > 5:
        return phone[:5] + "-" + phone[5:]
    return phone


def parse_cgpa(cgpa_str):
    """
    If the string contains '%', label it as 'Percentage: ...'
    Else 'CGPA: ...'
    If it's 'Not Found', keep it as 'Not Found'.
    """
    if not cgpa_str or cgpa_str.lower() == "not found":
        return "Not Found"
    if '%' in cgpa_str:
        return f"Percentage: {cgpa_str}"
    return f"CGPA: {cgpa_str}"


def clean_field(value):
    """Remove excessive whitespace & newlines."""
    if not isinstance(value, str):
        return value
    return " ".join(value.split())

def extract_full_details_gpt(text, retries=3, delay=10):
    """
    Uses GPT to parse multiple degrees, phone/emails, etc.
    We strongly request GPT not to omit phone/email details.
```

```
    """
    text_hash = get_text_hash(text)
    if text_hash in cache_dict:
        return cache_dict[text_hash]

    prompt = f"""
    You are an advanced AI system extracting comprehensive fields from the resume text.
    Do not omit or skip phone/email details even if repeated.
    Distinguish 'course' vs 'discipline' carefully:
      - 'course': "Bachelor of Engineering", "Master of Technology", "Diploma", etc.
      - 'discipline': the field, e.g. "Cyber Security", "CSE", "Electronics".
    Data to extract:
    1. Name
    2. PhoneNumbers: a list of all phone numbers found
    3. Emails: a list of all emails found
    4. Education: a list of dicts, each with:
       - institution
       - course
       - discipline
       - year (e.g., "2022" or "2022-24")
       - cgpa
    5. KeySkills: a list of strings
    6. Experience: a brief summary of professional experience
    7. Certifications: a list or summary of relevant certifications
    8. Projects: a list or summary of relevant projects
    9. AdditionalInsights: e.g., awards, volunteering, interests, languages
    Return valid JSON with EXACTLY these keys:
    {{
      "Name": "",
      "PhoneNumbers": [],
      "Emails": [],
      "Education": [],
      "KeySkills": [],
      "Experience": "",
      "Certifications": "",
      "Projects": "",
      "AdditionalInsights": ""
    }}
    If something is not found, fill with "Not Found" or empty lists.
    Resume:
    {text}
    """

    for attempt in range(retries):
        try:
            response = openai.ChatCompletion.create(
                model="gpt-3.5-turbo",  # or "gpt-4"
                messages=[{"role": "system", "content": prompt}],
                max_tokens=1200,
                temperature=0
            )
            content = response["choices"][0]["message"]["content"].strip()
            data = json.loads(content)

            # Clean top-level fields
            data["Name"] = clean_field(data["Name"])
            data["Experience"] = clean_field(data["Experience"])
            data["Certifications"] = clean_field(data["Certifications"])
            data["Projects"] = clean_field(data["Projects"])
            data["AdditionalInsights"] = clean_field(data["AdditionalInsights"])

            # Normalize phone
            cleaned_phones = []
            for p in data["PhoneNumbers"]:
                phone_fmt = format_phone_number(p)
                cleaned_phones.append(phone_fmt)
            data["PhoneNumbers"] = cleaned_phones

            # Clean emails
            data["Emails"] = [e.strip().lower() for e in data["Emails"]]

            # For each education entry, remove whitespace
            # Then fix discipline if GPT reversed them
            for edu in data["Education"]:
                for k in edu:
                    edu[k] = clean_field(edu[k])
                # If discipline is empty but course has typical discipline keywords
```

```
                if ((not edu["discipline"] or edu["discipline"].lower() == "not found")
                    and re.search(r"(cse|cyber|security|telecommunication|computer|electronics)", edu["course"].lower())):
                    edu["discipline"] = edu["course"]
                    edu["course"] = ""
                # Standardize CGPA or Percentage
                edu["cgpa"] = parse_cgpa(edu["cgpa"])

            # Clean each skill
            data["KeySkills"] = [clean_field(skill) for skill in data["KeySkills"]]

            cache_dict[text_hash] = data
            return data

        except openai.error.RateLimitError:
            print(f"Rate limit error. Retrying in {delay} seconds...")
            time.sleep(delay)
        except Exception as e:
            print(f"Error extracting (attempt {attempt+1}/{retries}): {e}")
            break

    # Fallback if fails
    fallback_data = {
      "Name": "Not Found",
      "PhoneNumbers": [],
      "Emails": [],
      "Education": [],
      "KeySkills": [],
      "Experience": "Not Found",
      "Certifications": "Not Found",
      "Projects": "Not Found",
      "AdditionalInsights": "Not Found"
    }
    cache_dict[text_hash] = fallback_data
    return fallback_data


def score_resume_gpt(text, job_description=None, retries=3, delay=10):
    """
    Scores:
     - GenAI_Score (1-3)
     - AI_ML_Score (1-3)
     - JobMatch_Score (1-10) if job_description is provided
    """
    if job_description:
        job_part = f"""
        Also evaluate how well this candidate matches the job description below on a scale of 1-10:
        {job_description}
        """
    else:
        job_part = ""

    prompt = f"""
    You are an AI assistant evaluating a resume. Provide three scores:
    1. Generative AI Experience (GenAI_Score): (1=Exposed, 2=Hands-on, 3=Advanced)
    2. AI/ML Experience (AI_ML_Score): (1=Exposed, 2=Hands-on, 3=Advanced)
    3. Job Match Score (JobMatch_Score) from 1-10,
       where 1=Poor fit, 10=Excellent fit. If no job description is provided, return 0 for this.

    Return valid JSON only:
    {{
       "GenAI_Score": <number>,
       "AI_ML_Score": <number>,
       "JobMatch_Score": <number>
    }}

    Resume Content:
    {text}

    {job_part}
    """

    for attempt in range(retries):
        try:
            response = openai.ChatCompletion.create(
                model="gpt-3.5-turbo",
                messages=[{"role": "system", "content": prompt}],
                max_tokens=300,
```

```
                temperature=0
            )
            result = response["choices"][0]["message"]["content"].strip()
            scores = json.loads(result)
            return scores
        except openai.error.RateLimitError:
            print(f"Rate limit error (scoring). Retrying in {delay} seconds...")
            time.sleep(delay)
        except Exception as e:
            print(f"Error scoring (attempt {attempt+1}/{retries}): {e}")
            break

    # Fallback
    return {
      "GenAI_Score": "Error",
      "AI_ML_Score": "Error",
      "JobMatch_Score": 0
    }


def multiline_enum(lst):
    """
    Return each string in a new line enumerated:
    1. item1
    2. item2
    """
    if not lst:
        return "Not Found"
    enumerated = [f"{i}. {val}" for i, val in enumerate(lst, start=1) if val and val.lower() != "not found"]
    return "\n".join(enumerated) if enumerated else "Not Found"

def multiline_text(text):
    """
    Splits text by lines, enumerates each line.
    """
    if not text or text.lower() == "not found":
        return "Not Found"
    lines = [ln.strip() for ln in text.split('\n') if ln.strip()]
    enumerated = [f"{i}. {val}" for i, val in enumerate(lines, start=1)]
    return "\n".join(enumerated) if enumerated else "Not Found"

def bulletify_supporting_info(certifications, projects, insights, experience):
    """
    Format the Supporting Information into a structured, multi-section bullet list:
    - Certifications
    - Projects
    - Additional Insights (Awards, Interests, Languages, Volunteering)
    - Experience
    """
    # Convert each to enumerated lines
    cert_lines = [line.strip() for line in certifications.split(',') if line.strip()] if certifications not in ["Not Found", ""] else []
    proj_lines = [line.strip() for line in projects.split(',') if line.strip()] if projects not in ["Not Found", ""] else []
    # We'll handle Additional Insights line by line as well.
    insight_lines = [line.strip() for line in insights.split(',') if line.strip()] if insights not in ["Not Found", ""] else []
    # Experience might be multi-line
    exp = multiline_text(experience)

    # Create final structure
    output = ""

    # Certifications
    output += "Certifications:\n"
    if cert_lines:
        for i, val in enumerate(cert_lines, start=1):
            output += f"{i}. {val}\n"
    else:
        output += "Not Found\n"

    # Projects
    output += "\nProjects:\n"
    if proj_lines:
        for i, val in enumerate(proj_lines, start=1):
            output += f"{i}. {val}\n"
    else:
        output += "Not Found\n"

    # Additional Insights
```

```
        output += "\nAdditional Insights:\n"
        if insight_lines:
            # Check if lines contain keywords like "Awards:", "Interests:" etc.
            # or just list them
            for i, val in enumerate(insight_lines, start=1):
                output += f"{i}. {val}\n"
        else:
            output += "Not Found\n"

        # Experience
        output += f"\nExperience:\n{exp}\n"

        return output.strip()

    def flatten_education(education_list):
        """
        Flatten education info into enumerated lines for each field
        (institution, course, discipline, year, cgpa).
        """
        institutions, courses, disciplines, years, cgpas = [], [], [], [], []

        for edu in education_list:
            institutions.append(edu.get("institution", "Not Found"))
            courses.append(edu.get("course", "Not Found"))
            disciplines.append(edu.get("discipline", "Not Found"))
            years.append(edu.get("year", "Not Found"))
            cgpas.append(edu.get("cgpa", "Not Found"))

        return {
            "University": multiline_enum(institutions),
            "Course": multiline_enum(courses),
            "Discipline": multiline_enum(disciplines),
            "YearOfStudy": multiline_enum(years),
            "CGPA": multiline_enum(cgpas)
        }


    def process_resume_queue(resume_queue, job_description=None):
        """
        Demonstrates queue-based approach for large-scale or advanced error handling.
        """
        results = []
        while not resume_queue.empty():
            resume_file = resume_queue.get()
            text = resume_texts[resume_file]

            # GPT Extraction
            details = extract_full_details_gpt(text)
            # GPT Scoring
            scores = score_resume_gpt(text, job_description=job_description)

            # Flatten education
            edu_info = flatten_education(details["Education"])

            # Prepare multi-line enumerations for phone/emails/skills
            phone_str = multiline_enum(details["PhoneNumbers"])
            email_str = multiline_enum(details["Emails"])
            skills_str = multiline_enum(details["KeySkills"])

            # Create structured "Supporting Information"
            structured_si = bulletify_supporting_info(
                details["Certifications"],
                details["Projects"],
                details["AdditionalInsights"],
                details["Experience"]
            )

            row = {
                "File Name": resume_file,
                "Name": details["Name"],
                "Contact details": f"Phone(s):\n{phone_str}\n\nEmail(s):\n{email_str}"
                "University": edu_info["University"],
                "Year of Study": edu_info["YearOfStudy"],
                "Course": edu_info["Course"],
                "Discipline": edu_info["Discipline"],
                "CGPA/Percentage": edu_info["CGPA"],
                "Key Skills": skills_str,
```

```
            "Gen AI Experience Score": scores["GenAI_Score"],
            "AI/ML Experience Score": scores["AI_ML_Score"],
            "Supporting Information": structured_si,
            "Job Match Score (1-10)": scores["JobMatch_Score"]
        }
        results.append(row)
        resume_queue.task_done()
```

```
# Example job description for match scoring
job_description = """
Looking for a Cybersecurity Specialist with strong AI/ML skills,
experience with generative models, and advanced knowledge of
blockchain, threat detection, and secure software development.
"""

resume_queue = queue.Queue()
for rfile in resumes:
    resume_queue.put(rfile)

processed_results = process_resume_queue(resume_queue, job_description=job_description)


df = pd.DataFrame(processed_results)
output_path = "/content/drive/MyDrive/Generative AI-Powered Resume Analyzer/Output/Final_Resume_Analysis.xlsx"
df.to_excel(output_path, index=False)
print(f"Enhanced results saved to: {output_path}")

df.head(3)
```

```
Enhanced results saved to: /content/drive/MyDrive/Generative AI-Powered Resume Analyzer/Output/Final_Resume_Analysis.xlsx
```

	File Name	Name	Contact details	University	Year of Study	Course	Discipline	CGPA/Percentage	Key
0	Swapnil_22MCES16.pdf	Swapnil Nandanwar	Phone(s):\n1. +91-9960140971\n2. +91-702030352...	1. Institute of Technology, Nirma University\n...	1. 2022-24\n2. 2017\n3. 2018\n4. 2014	1. Master of Technology\n2. Bachelor of Techno...	1. CSE - Cyber security\n2. Electronics & Tele...	1. CGPA: 7.62\n2. CGPA: 8.36\n3. Percentage: 6...	Cybersec Hac
1	Prathamesh Bute_Resume.pdf	Prathamesh Bute	Phone(s):\n1. +91\n\nEmail(s):\n1. prathamesh....	1. VIIT, Pune\n2. Govt. Polytechnic Nagpur	2. 2022-Present	1. Bachelor of Engineering (B.E.)\n2. Diploma	1. Electronics and Telecommunication Engineeri...	Not Found	1. Implement Imple
2	Manan_Patel_Resume.pdf	Manan Patel	Phone(s):\n1. +91\n\nEmail(s):\n1. emrpatel070...	1. Institute of Technology, Nirma University\n...	1. 2022-24\n2. 2019-22\n3. 2016-19	1. Master of Technology\n2. Bachelor of Engine...	1. Cyber Security\n2. Computer Engineering\n3....	1. CGPA: 7.96\n2. CGPA: 8.42\n3. CGPA: 9.37	1. Py Java\r C++\

Next steps: Generate code with df | View recommended plots | New interactive sheet